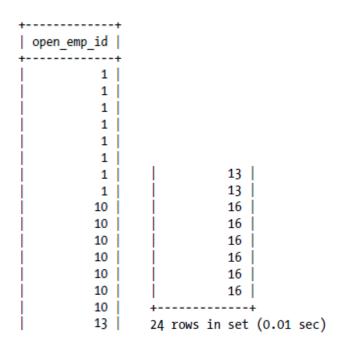
## Лекция 7. Группировка данных. Агрегатные функции

Данная лекция посвящена группировке и агрегированию данных, которые обеспечивают пользователям возможность работать с данными на более высоком уровне детализации, чем тот, с которым они хранятся в БД.

## Принципы группировки

Иногда необходимо выявить в данных некоторые тенденции, что требует от сервера некоторой подготовки данных, прежде чем можно будет получить искомые результаты. Например, вы отвечаете за операции в банке и хотели бы выяснить, сколько счетов открывает каждый операционист банка. Для просмотра необработанных данных можно было бы создать следующий простой запрос:

SELECT open\_emp\_id FROM account;



В таблице **account** всего 24 строки, поэтому относительно просто увидеть, что счета открывались четырьмя сотрудниками и что сотрудник с ID 16 открыл шесть счетов. Но для банка с десятками сотрудников и тысячами открываемых счетов данный подход оказался бы очень утомительным и подверженным серьезным ошибкам.

Вместо этого можно с помощью блока *group by* попросить сервер БД сгруппировать данные:

SELECT open\_emp\_id FROM account GROUP BY open\_emp\_id;

```
      1
      4
      строки: по одной строке для каждого отдельного значения столбца ореп_етр_id
```

Чтобы увидеть, сколько счетов открыл каждый сотрудник, в блоке *select* можно подсчитать количество строк в каждой группе с помощью *агрегатной* функции **count**():

SELECT open\_emp\_id, COUNT(\*) how\_many FROM account GROUP BY open emp\_id;

open_emp_id	how_many
1 10 13	8   7   3
16	6   +

Агрегатная функция **count**() подсчитывает количество строк в каждой группе, а 38e3do4ka (\*) предписывает серверу сосчитать все строки в группе.

При группировке может понадобиться отфильтровать из результирующего набора ненужные данные, опираясь на информацию групп данных, а не необработанных данных. Блок *group by* выполняется *после* вычисления блока *where*, поэтому условия фильтрации нельзя добавлять в блок *where*. Вместо этого можно поместить условия фильтрации группы в блок *having*.

SELECT open\_emp\_id, COUNT(\*) how\_many FROM account GROUP BY open\_emp\_id HAVING COUNT(\*) > 4;

open_emp_id	how_many
1   10   16	8     7
+	+

Группы, содержащие меньше 5 элементов, были отфильтрованы с помощью блока *having*, и теперь результирующий набор включает только сотрудников, открывших 5 или более счетов.

### Агрегатные функции

**Агрегатные функции** осуществляют определенную операцию над всеми строками группы. Хотя у всех серверов БД есть собственные наборы специализированных агрегатных функций, большинством из них реализованы следующие общие агрегатные функции:

- **Max**() возвращает максимальное значение из набора.
- **Min**() возвращает минимальное значение из набора.
- **Avg**() возвращает среднее значение набора.
- **Sum**() возвращает сумму значений из набора.
- Count() возвращает количество значений в наборе.

Вот запрос, использующий все обычные агрегатные функции для анализа доступных остатков (available balance) всех текущих счетов:

```
SELECT MAX(avail_balance) max_balance,
    MIN(avail_balance) min_balance,
    AVG(avail_balance) avg_balance,
    SUM(avail_balance) tot_balance,
    COUNT(*) num_accounts
FROM account
WHERE product_cd = 'CHK';
```

_	min_balance	avg_balance	tot_balance	num_accounts
38552.05	122.37	7300.800985	73008.01	

# Сравнение неявных и явных групп

В предыдущем примере все значения, возвращаемые по запросу, формируются агрегатной функцией, а сами агрегатные функции применяются

к группе строк, определенной условием фильтрации product\_cd = 'CHK'. Поскольку блок *group by* отсутствует, имеется единственная *неявная* группа (все возвращенные запросом строки).

Однако в большинстве случаев потребуется извлекать и другие столбцы, а не только сформированные агрегатными функциями. Если мы захотим выполнить эти же пять агрегатных функций для каждого типа счетов, а не только для счетов *СНК*, нам нужно будет *явно* задать способ группировки данных, используя блок *group by*:

```
SELECT product_cd,

MAX(avail_balance) max_balance,

MIN(avail_balance) min_balance,

AVG(avail_balance) avg_balance,

SUM(avail_balance) tot_balance,

COUNT(*) num_accts

FROM account

GROUP BY product_cd;
```

product_cd	max_balance	min_balance	avg_balance	tot_balance	num_accts
BUS	9345.55	0.00	4672.774902	9345.55	2
CD	10000.00	1500.00	4875.000000	19500.00	4
CHK	38552.05	122.37	7300.800985	73008.01	10
MM	9345.55	2212.50	5681.713216	17045.14	3
SAV	767.77	200.00	463.940002	1855.76	4
SBL	50000.00	50000.00	50000.000000	50000.00	1

Если имеется блок *group by*, сервер знает, что сначала надо сгруппировать строки с одинаковым значением в столбце  $product\_cd$ , а затем применить пять агрегатных функций к каждой из шести групп.

#### Подсчет уникальных значений

При использовании функции **count()** для определения числа элементов в каждой группе существует выбор: или пересчитать все элементы группы, или посчитать только *уникальные* (*distinct*) значения столбца из всех элементов группы. Рассмотрим, например, следующие данные, которыми представлены сотрудники, ответственные за открытие каждого счета:

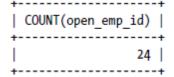
SELECT account\_id, open\_emp\_id FROM account ORDER BY open\_emp\_id;

account_id	open_emp_id
8	1
9	1
10	1
12	1
13	1
17	1
18	1
19	1
1	10
2	10
3	10
4	10
5	10
14	10
22	10
6	13
7	13
24	13
11	16
15	16
16	16
20	16
21	16
23	16
4 rows in set	· (0.00.505)

Как можно увидеть, все счета были открыты четырьмя разными сотрудниками с ID = 1, 10, 13 и 16.

Допустим, хочется подсчитать число открывших счета сотрудников – не вручную, а с помощью запроса. Если к столбцу *open\_emp\_id* применить функцию **count**(), увидим следующие результаты:

SELECT COUNT(open\_emp\_id)
FROM account;



**count(\*)** в данном случае даст тот же самый результат

Если требуется подсчитать количество *уникальных* значений в группе, а не просто пересчитать число строк в ней, нужно указать ключевое слово *distinct*:

SELECT COUNT(DISTINCT open\_emp\_id) FROM account;

```
COUNT(DISTINCT open_emp_id) |
```

#### Использование выражений

В качестве аргументов агрегатных функций вы можете использовать не только столбцы, но и созданные вами выражения. Например, требуется найти максимальное значение отложенных вкладов по всем счетам, которое вычисляется путем вычитания доступного остатка из отложенного остатка:

SELECT MAX(pending\_balance - avail\_balance) max\_uncleared FROM account;

Применяемые в качестве аргументов агрегатных функций выражения могут быть настолько сложными, насколько это нужно, и возвращать число, строку или дату.

### Обработка значений Null

При агрегировании так же, как и при вычислении любого численного выражения, всегда следует учитывать влияние значения *null* на результат вычисления. Пусть имеется таблица **number\_tbl** с единственным столбцом *val*, хранящим числовые значения {1, 3, 5}.

Рассмотрим запрос, применяющий пять агрегатных функций к этому набору чисел:

```
SELECT COUNT(*) num_rows,
COUNT(val) num_vals,
SUM(val) total,
MAX(val) max_val,
AVG(val) avg_val
FROM number_tbl;
```

num_rows	num_vals	total	max_val	avg_val
3	3	9	5	3.0000

Функции count(\*) и count(val) возвращают одинаковое значение 3.

Теперь добавим в таблицу **number\_tbl** значение *null* и выполним запрос еще раз:

num_rows	num_vals	total	max_val	avg_val
4	3	9	5	3.0000

Даже при добавлении в таблицу значения null функции sum(), max(), avg() и count(val) возвращают те же значения; это означает, что они игнорируют все встречающиеся значения null. Функция count(\*) теперь возвращает значение 4, что является правильным, поскольку в таблице  $number\_tbl$  стало четыре строки.

Таким образом, функция **count**(\*) считает строки и поэтому не подвержена влиянию значений null, содержащихся в строке. А функция **count**(val) считает значения в столбце val, пропуская все встречающиеся значения null.

## Литература:

1. Алан Бьюли. Изучаем SQL: пер. с англ. – СПб-М.: Символ, O'Reilly, 2007. - 310 с.